National College of Ireland

Implementation and Evaluation of a Zero Trust Architecture to Mitigate Multitenant Data Leakage In .Net Core Azure Applications

MSc Research Project
Cloud Computing

Nidhi Anandan
Student ID: 23286873

School of Computing
National College of Ireland

Supervisor: Dr. Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Nidhi Anandan
Student ID:	23286873
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr. Shivani Jaswal
Submission Due Date:	11/12/2025
Project Title:	Implementation and Evaluation of a Zero Trust Architecture to Mitigate Multitenant Data Leakage In .Net Core Azure Applications
Word Count:	8000+
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nidhi Anandan
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

MSc Research Project

Implementation and Evaluation of a Zero Trust Architecture to Mitigate Multitenant Data Leakage In .Net Core Azure Applications

Your Number	Name/StudentCourse	Date
Nidhi Anandan/ 23286873	Msc. In Cloud Computing	11/12/2025

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

[Place evidence here]

Additional Evidence:

[Place evidence here]

Implementation and Evaluation of a Zero Trust Architecture to Mitigate Multitenant Data Leakage In .Net Core Azure Applications

Nidhi Anandan
23286873

Abstract

Zero Trust has become an emerging security architecture for cloud native applications, but still many SaaS applications depend on role based access control or simple tenant checks, which are vulnerable to misconfigurations and data leakage across tenants. This project designs and implements a zero trust middleware for .Net Core based Azure Function API's that will validate every API request and make tenant aware decision using a dynamic trust score calculation. This middleware will intercept every HTTP request derives request context details such as ip address, geolocation, tenant ID and access time then apply a configurable rule penalty to compute a numeric trust score for the request. Every decision is based on trust score to either allow or deny the request, this is done before executing any business logic of the API, also a Admin dashboard has been developed in .Net Core MVC that will allow the admin to setup the zero trust policy configurations and also configure which type of data can one particular tenant can access from the Azure Cosmos database, all the configurations are stored in Azure App configuration. We have also built simulated test requests using Grafana cloud K6 scripts that will help to evaluate the performance and security effectiveness under different work loads, this performance test can be triggered directly from the admin dashboard and the performance metrics can be viewed in Grafana dashboard. The results of the middleware shows that it can block low trust requests by adding a very low latency and keeping the throughput stable, this suggests that this middleware is a practical solution to reduce the risk of multitenant data leakage in .Net Core Azure applications which is light weight without causing high performance cost and making the system too complex. We have also published this middleware as a NuGet package which can be easily reused by other developers in their .Net Core Applications.

Keywords— Cloud Security, Zero Trust Architecture, Multitenancy Data Leakage, Azure Cloud, .Net Core, Azure Function API

1 Introduction

In this present era of cloud native software and distributed SaaS systems, multi tenancy has been a important and crucial architecture. It allows efficient resource sharing, cost effectiveness, scalability which allows different tenants to share cloud resources or services. However by doing this there is a new security issue that is arising called as multitenant

data leakage. This occurs when one tenant gets unauthorized access to data belonging to the other. Either because of weak identity control or misconfigured application logic. Cloud service providers such as Microsoft Azure have secured design, but still they rely on Developers for correct and secured implementation practices. According to Cloud Security Alliance (CSA) and OWASP OWASP Foundation (n.d.), applications that are designed for multi tenants must enforce strict tenant isolation and extend from static configurations to dynamic policy based enforcement mechanisms at the application layer.

This research project focuses particularly for .Net Core applications hosted on Microsoft Azure, where multiple Azure Entra ID groups have been used to simulate multiple tenants. While Azure offers identity level access boundaries but applications fails to implement dynamic data level access enforcement. The main objective of this project is to design, implement and evaluate a Zero trust architecture middleware that will evaluate multitenant data leakage through real time scoring evaluation at the API layer. This solution not only emphasizes identity based control system but also context aware decision making parameters like IP address, geo location and access time, where each request is validated individually before giving access to a shared data.

This zero trust architecture has been implemented like a middleware in .Net Core applications through Azure functions. This can be configured through a dedicated admin dashboard that is built using Asp .Net Core MVC, which stores the zero trust parameters in Azure App configuration. Trust score is calculated using weighted rule model and responses are determined using thresholds like low, medium and high trust levels. Depending on this score a request may result in allow, allow-read or deny action.

To validate the effectiveness of this proposed architecture the .Net Core API's that were developed in Azure functions were subjected to rigorous performance testing using K6 cloud scripts under different loads. This test suite has both positive and negative access patterns. This will generate metrics like request counts, latency, trust score breakdown, success or failure rates, decision types. This metrics are visualized in Grafana cloud dashboards which will allow the users to analyze the performance of the middleware. An important implementation detail is that this K6 tests can we triggered dynamically from the admin interface by selecting the number of API requests and a Github actions workflow is invoked to run the performance testing.

The key innovation of this project is a light weight dynamic policy enforcement at runtime, which is more flexible when compared to static access rules. While traditional models use hard coded claims or role based access controls. This proposed solution implements a scoring mechanism that will adapt to the request's context which allows dynamic policy enforcement without the need of redeploying of applications. Furthermore this middle ware is packaged as NuGet package, which allows the reuse across different .Net Core based SaaS applications which ensures secure multitenant isolation across different deployment models.

1.1 Research Question

To what extent can Zero Trust Architecture mitigate multitenant data leakage risks in shared cloud databases in .NET Core applications?

1.2 Motivation

In modern day cloud based Software has a Service (SaaS) platforms supports multiple tenants to share infrastructures and this is a common architectural pattern. Azure Entra ID plays a key role in enabling identity and access across environments. However this can manage user authentication and assign group based access, it does not offer deep context aware security at the data layer. In multitenant systems, this might create a risk where misconfigurations in access rights or any gaps in business logic would allow a user from one tenant to accidentally or maliciously access the other tenants data. This project is motivated by the need to address this critical gap by enforcing Zero Trust Architecture (ZTA) principles directly within the data access layer of shared .Net Core application hosted on Azure.

1.3 Ethics Consideration

This study does not include human subjects or private/public datasets, as per Table 1.

Table 1: Declaration of Ethics Consideration Table

Ethics Consideration	**Response**
This project involves human participants	Yes / No
The project makes use of secondary dataset(s) created by the researcher	Yes / No
The project makes use of public secondary dataset(s)	Yes / No
The project makes use of non-public secondary dataset(s)	Yes / No
Approval letter from non-public secondary dataset(s) owner received	Yes / No

1.4 Research Objectives

- Design and implement a lightweight Zero Trust .Net Core middleware that will validate every API request which enforces fine grained tenant specific access policies at the API data access layer.
- Evaluate the middleware by demonstrating the ability to differentiate between compliant and non compliant requests with configured policies with minimal false positives and false negatives.
- Evaluate the performance of the middleware for varying work load to determine whether the middleware enforces acceptable latency for production level multitenant applications.
- Demonstrate the practical feasibility of implementing the middleware in Azure functions and validate its compatibility in serverless executions.

1.5 Structure of Work

This project report begins with an introduction section that highlights the background of the study, research question, motivation and overall objectives of the study. In section 2 a

review of existing literature on Zero Trust Architecture, multitenant security challenges has been discussed. Section 3 outlines the research methodology which includes the overall development strategy, system design, performance testing, workflow and evaluation techniques. Section 4 provides in detailed design specification of this proposed solution, which discuses the architecture design, tenant isolation model, trust scoring mechanism and integration of various Azure services. Section 5 discusses the implementation phases like development of .Net Core web application (ZTA configuration UI), Azure functions, Azure app configuration, Cosmos database, zero trust middle ware along with it's scoring mechanism. In Section 6 the system is evaluated by a series of performance tests that is executed through github actions and K6 cloud scripts. This section also analyzes the performance of the middleware by comparing response latency, trust score outcomes for both positive and negative requests, which also allows the users to view the performance metrics in Grafana cloud. Finally Section 7 concludes the report with a summary of key findings, limitations and future enhancements of the project.

2 Related Work

2.1 Cloud and Multitenant Security Background

Cloud environments provide scalability, flexibility and cost efficiency but a new security challenge is introduced because of shared infrastructure. Studies show that threats like data breaches, insider misuse, insecure interfaces which denotes that traditional perimeter based security methods are inadequate to resolve this issue especially for cloud environments (Krishnappa et al.; 2024).

Cloud security experts often recommend to use several protection methods working together. Some of them are implementing strong user access controls, encryption of data and compliance checking. While these methods help but they also have a major limitation as they treat all customers as same and focus on only protecting the cloud infrastructure. They do not focus on protecting different customers sharing the same database or API services. This is one of the big problem in SaaS platforms where even one mistake can expose tenant's data to another tenant (Hayat et al.; 2024).

2.2 Multitenant Architecture and Isolation Techniques

Multitenant systems often share computers, storage and software across many customers. This helps to reduce cost and also improves efficiency however, if the isolation fails it will affect all the customers at once. Researchers have implemented different architecture designs to handle multitenant systems some use separate databases for each customer, others one database with customer identifier and some also uses virtual machines for isolation, but each method has some trade offs between speed, cost and security strength (Chippagiri; 2025).

Some researchers tried using string math based encryption techniques called as FHE and SMPC to protect data, this method is very secure. It can prevent protect data even from hackers that use advanced attacks. However it is very slow and does not scale well. When we add more customers to the data the system becomes very slow. It take tens of seconds to respond, which is not suitable for real time business applications(Ponmalar et al.; 2024).

Newer research based on zero trust data lakes shows a better approach. These systems uses splitting of data, customer specific encryption keys and end to end encryption. Test results shows that they work very well. They protect 100 percent of data adding only 8 percent extra delay. They can handle large number of data more efficiently. However the system is not designed for large data analytics systems like Hadoop or snowflake. They are also not suitable for SaaS applications that need quick business transactions(Anuganti; n.d.).

2.3 Zero Trust Architecture, Principles and Multi Cloud Perspectives

The idea of zero trust comes from the problems because of tradition security. Old security assumes everything inside the computer network is safe but in cloud computing this is wrong. Zero Trust says never trust anything always verify everything. Every user, device and every request needs to be checked no matter where the request comes from. There are different ways to build zero trust systems. Some of them use special agents on devices, others without agents also called as agentless. Some use software defined perimeters. However, all this will face some real challenges. These may not work well with the old systems and can make the system slower (Mavroudis; 2024).

Some companies have also implemented zero trust systems in different ways. One of them updated an existing AWS application by adding strong security checks for https connections and user verification. This improved the security without the need of any big code change in the application (Wang et al.; 2024). Another company built a zero trust system using Kubernetes a container orchestration platform they used JWT tokens, mutual TLS and network segmentation. Their test results shows that they were able to prevent unauthorized access and kept response time in a acceptable range (Viswanathan et al.; 2024). Other research looked at implementing zero trust system across different cloud service providers like AWS, Azure and Google cloud. This requires to have the same security policies across the platforms. However network and infrastructure layers got most of the focus (kumar Polinati; 2025). Some studies also examines zero trust architecture especially across multiple cloud environments but implement consistent identity and access policies across different cloud service providers (Rehan; 2022).

2.4 AI Driven Zero Trust, Trust Scoring Models and Threat Handling

Some researchers have combined artificial intelligence with zero trust systems they are also called as AIZT which stands for adaptive intelligence for zero trust. The main idea is to build a trust score of each request using machine learning. The score is determined and calculated based on different factors like device status, location and activity patterns. If the score is high the system will allow access if the score is low it will deny the access to the system. This system will learn and improve over time. Test results shows that this worked better when compared to fixed rules. This system identified most of the attacks and also has less false alarms (Obaze et al.; 2025).

Another system uses deep learning specifically LSTM and auto encoders that will help to spot unusual behavior. Test results shows that it blocked 96 percent of unauthorized requests and also has a accuracy rate of 92 percent. Some extra delay is added and it can also handle thousands of users at once (Mehata; 2025). Beyond from just access

control AI is used for also threat hunting in cloud based systems. This means using of machine learning to identify suspicious activity and respond to attacks faster than humans can (Denis et al.; 2025). These shows that using continuous risk assessment and smart trust scores can improve overall security in cloud environments. However, most of this AI systems focus on network level decision making for example which servers can talk to each other or account level decisions like which user accounts can login. They rarely rarely provide tools that can work directly inside the application at a API level because that is where the business data is accessed. This is where the gap exists (Denis et al.; 2025).

2.5 Azure, SaaS and API Specific security Patterns

Many researches have implemented many security pattern libraries especially for SaaS applications that are running on Azure. These patterns covers how can we isolate customers, exchange security tokens in a secured way, role based access controls and also about compliance checking. Real world SaaS companies are offering CRM tools, API Health management software to identify patterns successfully (Balasubramanian; n.d.).

Other researchers have combined zero trust architecture with risk management frameworks like NIST 800-37 which suggests that SaaS customers should not trust the cloud service provider and they need to actively verify that their data is protected correctly by the cloud service provider (Vesga; 2024). Another important area is API security, Api's are the entry gates to the SaaS applications data and they need to be treated as a potential dangerous entry points, a good secured API should validate every request by enforcing least privilege by allowing minimal permissions and checking for any suspicious behavior continuously and stopping it if there is any unusual activity (Steingartner et al.; 2024).

All these papers provide good guidance and best practices but they are mostly based on general principles and patterns none of them actually build and test a real time working system that enforces Zero trust rules at the API level and their performance impact.

2.6 Synthesis and Research Gap

When we look at all the previous related work several clear patterns appear, cloud security experts agree that cloud based multitenant systems are risky for data leakage across tenants and need strong security controls (Krishnappa et al.; 2024) (Hayat et al.; 2024). Strong encryption and isolation techniques are available but they are slow which will affect the real time processing of data which is impractical for business use (Ponmalar et al.; 2024) (Anuganti; n.d.). Companies have successfully proved zero trust architecture principles and implemented them in in AWS, Kubernetes across different cloud service providers (Mavroudis; 2024)(Wang et al.; 2024)(Viswanathan et al.; 2024)(Obaze et al.; 2025) Zero trust has also been implemented in small resource limited environments for an example in rural health clinics and small business, this shows that this principle is flexible(Müller; 2025).

Despite all this research works no one has created a practical, reusable light weight and low cost zero trust middle ware for .Net Core applications running on Azure functions and another important feature is this system allows the administrators to set custom rules for each tenants for every API request based on tenant ID, IP address, location, access time which makes real time decision like allow, read only and deny access.

All the existing studies focuses on either network or infrastructure layers instead of

application or API layers and they are mostly designed on AWS or Kubernetes instead of .Net Core and Azure, some provide only theoretical suggestions without working code, some use very strong encryption methods but they are slow.

SaaS companies need practical implemented solution and that can be easily integrated in their applications. Most modern day applications are built using .Net Core and Azure as both are owned by Microsoft and work seamlessly together. Companies might need a system focusing on .Net Core and Azure applications which allows the configuration per tenant and does not have much performance issue. This research project fills this gap by building such a system.

Table 2: Summary of Related Work and Gap Analysis

Reference	**Main topic / contribution**	**Approach & platform**	**Strengths**	**Limitations vs this project**
(Krishnappa et al.; 2024) , (Hayat et al.; 2024)	General cloud, multitenant security challenges	Literature Analysis of threats, IAM encryption and IDS in cloud environments	clear motivation for cloud security to handle multitenant risks, introduces layered defense concepts	High level, no solid zero trust middle ware or tenant level data level isolation for specific stacks.
(Chippagiri; 2025)	Security framework for multitenant cloud architectures	Survey of multitenant patterns and security models	Addresses tenant isolation, data segmentation and compliance	Very conceptual, does not implement trust score validation at application or API layer.
(Ponmalar et al.; 2024)	FHE, SMPC encryption for securing multitenant data	Cryptographic computation for encryption of data with strong isolation	Very string protection against cryptographic attacks	High latency, poor scalability for large data, and not suitable for lightweight SaaS middlewares.
(Mavroudis; 2024),(Wang et al.; 2024)	Zero Trust principles and AWS implementation	Survey of ZTA Architectures and Transparent shaping an AWS web app	Practical implementation of ZTA by migrating from perimeter to ZTA and strong conceptual basis	Focuses on network or web level not on data layer and also for Azure and .Net

Reference	Main topic / contribution	Approach & platform	Strengths	Limitations vs this project
(kumar Polinati; 2025)	ZTA in multi cloud and Kubernetes	Frameworks using RBAC, mTLS, JWT and micro segmentation	Demonstrate ZTA in distributed work loads by reducing lateral moment	Platform focuses on Kubernetes not on Azure functions and no tenant configurable scoring model.
(Steingartner et al.; 2024)	ZTA API security	Uses NIST ZTA principles for API controls and real world API breaches	strong justification for each request and least privilege access for API's	Conceptual and provides guidance only no practical implementation or performance evaluation.
(Denis et al.; 2025)	AI based threat hunting in private clouds	Review of ML models and integration them with SIEM and monitoring	Shows AI improves anomaly detection	Focuses more on SOC or monitoring layer not for API access decisions or tenant level data isolation.
(Müller; 2025)	ZT hybrid backup for rural health clinics or small businesses	Zero trust, local disk disaster recovery and edge AI detection.	Practical implementation of ZTA with low cost and resilience benefits	Domain specific not focused on SaaS applications or reusable middle ware components.
(Anuganti; n.d.)	Zero Trust multitenant data lake architecture	Tenant scoped keys, end to end encrytion and micro segmentation	Acheives 99.97 percent isolation effectiveness with less than 8 percent query overhead.	Targets on data lake/ analytics workloads not for SaaS API's

Reference	Main topic / contribution	Approach & platform	Strengths	Limitations vs this project
(Obaze et al.; 2025)	AIZT AI based zero trust scoring system	Mathematical model is used to combine classifiers and RL to calculate trust score	very conceptual to dynamic trust scoring but improved detection of risks	General cloud setting not for Azure and light weight reusable .Net implementation

3 Research Methodology

This chapter discusses the research methodology that was used to develop and test the zero trust middleware for multitenant SaaS applications. This research follows a design science approach, where the main goal is to build a working system practically and test its effectiveness by various experiments. The methodology is divided into four phases literature review, architecture design, system design and implementation and evaluation. This chapter outlines the research approach, experimental design and success criteria that is used to measure the security and performance of the middleware.

3.1 Research Approach

This research is divided into four phases first we understand the existing literature review and understand what others have done. Second we design a architecture for a Zero trust .Net Core middleware. Third we build a system using .Net Core and Azure resources. Last we test the system to measure both the performance and security of the middleware.

The entire process is documented so that users can implement the same in their environments and run the experiments using their own Azure subscription. This shows that our research are repeatable and verifiable. This research methods uses standard practices for software engineering. Where the goal is to build a system and prove it works through experiments.

3.2 System Design and Architecture

This system is built focusing on SaaS environment. It simulates a real SaaS application where multiple customers or tenants share the same cloud infrastructure and admin control the data access of each tenant.

A .Net Core web application has been developed which acts as an admin portal. This is where admin can manage tenants setup zero trust security settings. Execute Azure function Api's in real time and run performance tests. The backend Api's are developed using Azure functions which are serverless compute resources. We have used Azure Entra ID (Azure Active directory) groups to simulate multiple tenants. Each group represents different tenants and we can control which data they can access.

Tenant specific settings are stored in Azure App configuration, this is a Azure cloud configuration service that lets us to read or update the settings in real time. So if a admin

changes the security rules we can read them immediately without the need of redeploying or stopping the system.

A custom middleware has been developed and integrated in Azure functions API project. This will run on every incoming request. It reads who is making the request and which tenant they belong and load the corresponding settings from app configuration. Then it collects the contextual information like client's IP address, geolocation and time of request from the header of the API request. It will then evaluate if this contextual information matches the admin configured security rules. It will calculate a trust score by subtracting penalties for any rules that are violated. Finally once the score is calculated it compares this against the admin configured thresholds and make a decision like allow access, deny access completely or read only access.

This middleware is also packaged as a reusable Nuget Component. Nuget is a package manager for .Net. which means that other SaaS developers can easily reuse this middleware in their .Net Core applications with minimal code changes. This aligns with the modular design principles suggested and discussed by researchers (Viswanathan et al.; 2024).

3.3 Experimental Design and Data Collection

The evaluation compared two versions of Azure function API's, version one which is the base line Api without the middleware, and the other version with the enhanced API with full zero trust middleware. We define test scenarios where K6 will generate many virtual users over a period of time. K6 is a tool that will make requests to API endpoint we create two different types of requests positive traffic (50 percent of requests) and negative traffic(50 percent of request) that intentionally violate atleast one of the rule (IP address or access outside allowed times) so that the access is denied so that we can assert the correctness of the middleware. We can change the number of API requests (1000 to 5000) from the dashboard to run performance test and view the test results in Grafana cloud dashboard. During each test run the system logs data for every request like average trust score, access decision, Http status code, response latency. This is collected for baseline and enhanced version of API.

The main goal is whether this Zero trust middleware can meaningfully reduce the risk of unauthorized cross customer data access while keeping the latency (slowdown) within acceptable limits ideally between less than 200 to 300 milliseconds for each request which is suitable for enterprise level API's (Mavroudis; 2024).

3.4 Success Criteria and Measurement

Success Criteria is measured using three main indicators

- Security effectiveness Atleast above 95 percent of negative requests must be blocked or restricted.
- False positive rate Less than 5 percent of legitimate requests should not be incorrectly denied.
- performance impact The middleware must not add more than 200 to 300 milliseconds of latency for each request.

All the test results will be documented along with their responses time, success or denial rate, trust score distribution and comparison charts of latency between base line API

and the enhanced API will be available in Grafana cloud. This will allow the researchers to verify our findings and can adapt this methodology to their own environments.

Furthermore these metrics will provide a standardized framework for further studies, by understanding the security, accuracy and latency other researchers can contribute alternative algorithms or architectures against our results. This ensures that this contribution is not isolated but provides a reproducible baseline for broader field of cloud security research.

4 Design Specification

In this chapter we will discuss the design of the Zero trust middleware and the corresponding main SaaS components. The main aim is to analyze how this components interact with each other and how an API request is evaluated before it's allowed to access the tenants data. Implementation details will be discussed in the Implementation chapter.

4.1 High Level Architecture Overview

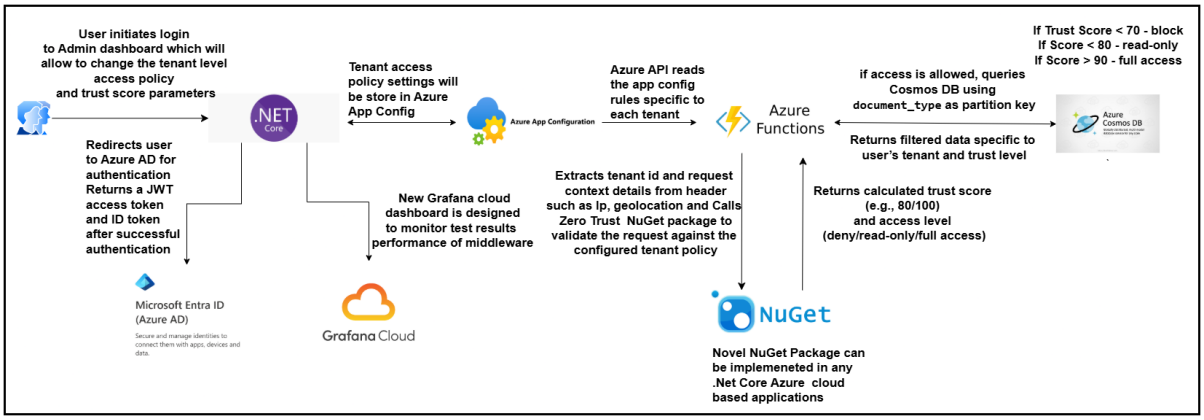

Figure 1: High level Architecture of the Zero Trust Multitenant System.

This solution has been developed in .Net Core using Azure cloud resources. The system's architecture consists of the following main components.

- Admin Portal (.Net Core 8.0 MVC Web Application) - This is the web application where admin can manage the tenants and their data access using the partition keys of the database. Also to configure zero trust rules such as IP ranges, location and access times that are used to determine the trust score.
- Azure Entra ID (Active Directory) - Azure Entra ID is used to authenticate the users into the web application.
- Azure Function API's - Serverless Http API endpoints that represents the business logic and are exposed to tenants for data access.
- Azure App Configuration - Azure App configuration stores tenant specific zero trust policies that are configured by admin and it is read by Function Api in run time.

- Zero Trust Middleware - Custom middleware that is running in Azure Functions project that evaluates every incoming API request and determines whether to allow, deny or read only access.
- Grafana Monitoring and Performance Tests - Grafana K6 Scripts are used to generate synthetic load against the API's to determine the performance of the API's and the performance metrics are visualized in Grafana dashboard.

4.2 Components Responsibilities

In this sub section we will discuss the clear responsibility about each component in detail.

- **Admin Portal (.NET Core 8.0 MVC Web Application)**

  This is a administrator dashboard where admin can edit tenant details and what data each tenant can access from the database based on the partition keys. The admin can also configure Zero trust rules per tenant, execute Function API's within the web page and trigger performance testing of API's by selecting the number of API calls. This dashboard does not perform access control checks but writes the configuration values to Azure app configuration so that the middleware can read that values in Azure functon API and enforce the zero trust settings and calculate trust score of each request based on the configuration values at runtime. This allows to keep the security logic at one place.

- **Azure Entra Id (Active Directory)**

  Azure Entra ID has been used to authenticate users into the admin dashboard. It issues JWT Tokens once the user is successfully authenticated which has critical identity information like users object ID, tenant ID and group memberships. Since this research project uses personal account there was a limitation to create multiple organizational tenants, to handle this two security groups have been created to represent Tenant A and Tenant B and test users have been created and added to the groups to simulate multiple tenants. The middleware extracts group memberships from the issued JWT token which helps to determine which tenant the user belongs to which enables multitenant isolation without the need of separate Azure AD tenants.

- **Azure Function API's**

  The Azure Function API expose serverless HTTP endpoints that handles the business logic and they are exposed to different tenants for data access. After successful authentication the request passes through the middleware before even it even executes any business logic, this ensures that regardless of the authentication of the request they are subjected to context aware trust evaluation before accessing the data.

- **Azure App Configuration**

  Azure App configuration stores all the tenant related zero trust policies in key value pair which are dynamically retrieved based on tenant id during run time for each request without the need of any code changes or code deployments. This allows the administrators to dynamically update the trust policies and threshold limit through admin portal and the API will immediately apply the updated setting to subsequent

requests. Using cloud based configuration instead of hard coded values is critical for multitenant SaaS system where the policies are tenant specific and frequently updated.

- **Azure Cosmos Database**

  Azure cosmos database is one of the widely used no SQL database that is offered by Azure, for this project we have used this as a multitenant database where each tenant data is stored and partitioned by document type (type of record). The Zero trust enforces tenant isolation at data layer which ensures that each tenant can access specific document types that are configured by the administrator.

- **Grafana Monitoring and Performance Tests**

  Grafana cloud is used to monitor the behavior and performance of the Zero trust middleware during testing and evaluation. The middleware will log all the latency metrics and responses times with and without the middleware number of successful and failed API requests also average trust scores of successful and failed requests will be logged in Grafana cloud. K6 scripts are used to generate simulated synthetic API requests for testing which simulates both positive and negative API requests where 50 percent are positive they satisfy the zero trust rules and 50 percent negative tests that intentionally violates the rules this 50:50 percent rules helps us to determine the correctness of the middleware. This performance metrics will be available in the newly developed Grafana dashboard. This monitoring is used only for testing and evaluation purposes and does not affect the runtime behavior of the production version.

4.3 Key Design Considerations

Several design decisions supports the practical implementation of this system in real world SaaS applications.

- Reusability - Implementing the middleware as a Nuget package will allow developers to reuse them in multiple .Net Core Projects.

- Light Weight Processing - This trust score calculation is implemented using deterministic rule evaluation and arithmetic operations instead of heavy computation or machine learning models which keeps the latency very low within 200-300 ms target defined in methodology.

- Low Coupling - The admin portal will communicate through Azure app configuration instead of direct service calls which makes sure each part can work independently.

- Cost Optimization - Using Serverless components such as Azure Functions and managed services reduces the running cost very low for small and medium sized SaaS providers.

5 Implementation

5.1 Admin Portal Dashboard

The admin portal is the dashboard where the administrators can define the zero trust policies for each tenant. Before any API requests the admin must configure the settings that the middleware will enforce. Admin can select from the list of Azure Entra groups which represents different tenants and they can specify which data partition they can access from the database such as Task or Task Assignment for an example. There are different types of rules like IP Address, geolocation or access time, an enabled checkbox indicates whether the rule is active and must be enforced by the middleware. The deduct input field denotes the the number of points that must be deducted out of 100 if the corresponding rule is violated. Admin can also configure trust score thresholds that are mapped directly to access actions like deny, allow read only, allow write access as shown in fig. 2. Requests that fail below the defined the threshold will be blocked, while those request that meet the medium or high trust scores gets the read only or full access respectively. This mechanism ensures that the trust score clearly defines the level of privilege granted. Finally a save configuration button will save the details to Azure App configuration for real time enforcement without restarts.

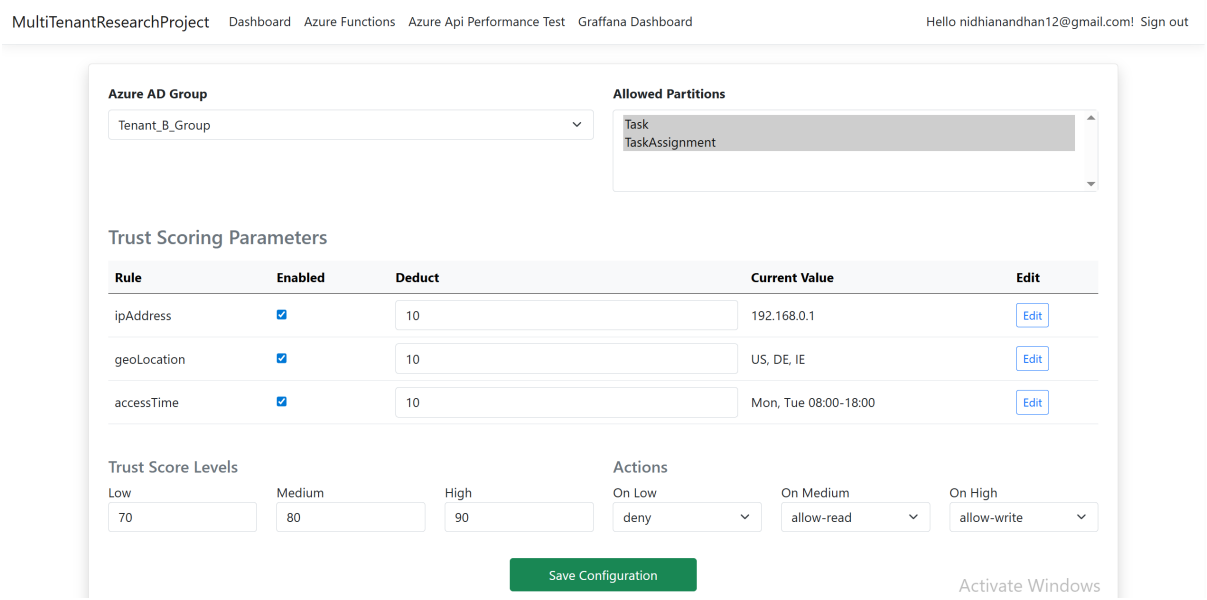

Figure 2: Admin Zero Trust Configuration Dashboard.

5.2 Swagger API Interface for API Testing

The Swagger interface is an interactive platform where users can execute and test the Azure Function API's in real time. Once the admin configures the security rules they can test the API if the rules are evaluated correctly by the middleware. Only Azure AD authenticated users can access this API's. Deevelopers can select the API endpoint from the avaliable list of API's, they can click on the try it out button and add the required parameters in the headers like API Key, Ip address, TenantID and then click on execute button. The middleware will process this request against the configured rules by admin by

calculating the trust score and making an access decision. The response will be displayed in swagger screen along with business data, middleware meta along with decision type, trust score and also information about the parameter responsible for trust score deduction as shown in fig. 3.

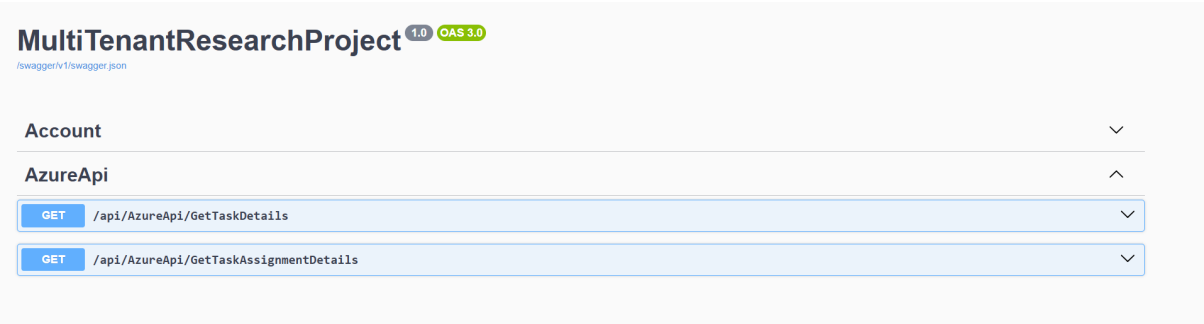

Figure 3: Swagger API Interface for API Testing.

5.3 API Performance Testing Environment with Test Configuration

This simulation testing web page will provide a dedicated space to users to test the performance and correctness before deploying it to production. The Admin as shown in fig. 4 can select the Api and number of API request from the range of 1000 to 5000 requests. Once the user clicks the Run test button the testing process will start by triggering a K6 cloud script. As this application is hosted on Azure App service its a sandboxed and does not allow running custom scripts to over come this we had to create a github action that orchestrates automated performance testing. We will make a Api call to this github action along with corresponding parameters like API and the number of API requests. Throughout the execution K6 will measure all the critical performance metrics which will be deployed directly to Grafana dashboard.

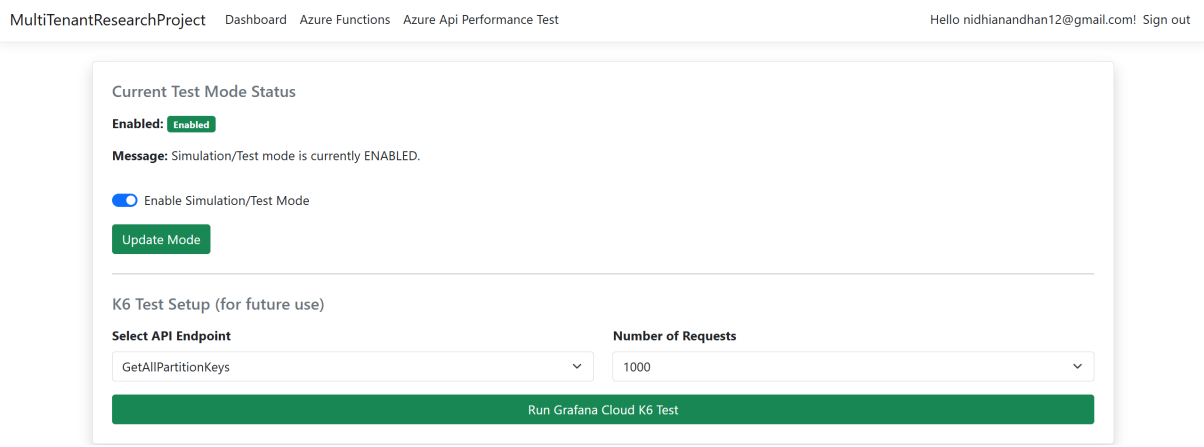

Figure 4: API Performance Testing Environment with Test Configuration.

5.4 Grafana Dashboard for Test Results Visualization and Analysis

The Grafana dashboard has been designed to provide a comprehensive visualization platform where all the K6 load and performance testing metrics are displayed to analyze and interpret the performance or effectiveness of the middleware. After the testing is done the results are directly exported to Grafana from the github actions where they are ingested into dashboard queries and visualizations. It displays multiple metrics like latency, time comparisons of response time between base line API and middleware enabled API's revealing the performance overhead introduced because of the enforcement of the middleware. The number of total requests which include total number of successful HTTP 200(Success) and denied requests HTTP 403(Deny) is also displayed. To validate the correctness of the middleware the 50 percent positive and 50 percent negative requests out of the total requests needs to be logged. The dashboard also displays the average trust score distribution responsible for each decisions as shown in fig. 5. These metrics demonstrate that the Zero trust middleware enforces successfully blocks negative requests maintaining performance within acceptable limits.

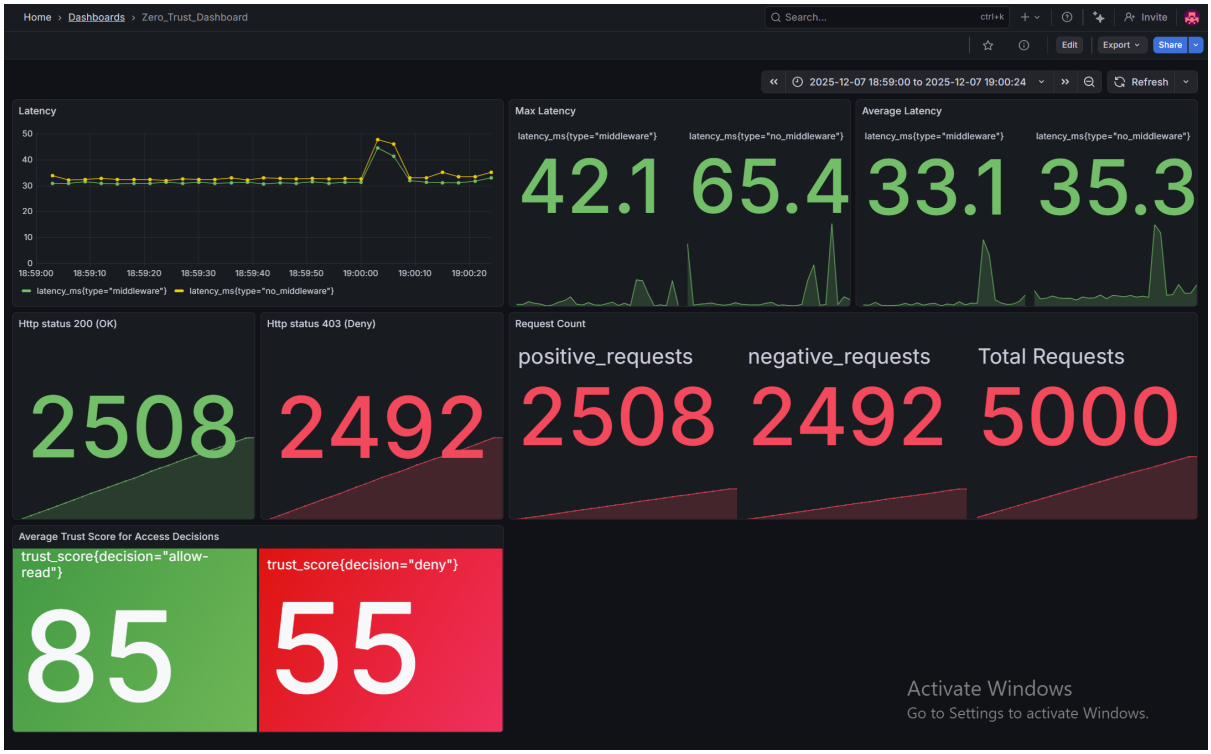

Figure 5: Grafana Dashboard for Test Results Visualization and Analysis.

6 Evaluation

In this section we will be discussing the evaluation of the middleware by two important perspectives such as security correctness and performance by increasing the request volume to the API's the main objective is to show the middleware enforces tenant rules while keeping the performance and error rates under control as the number of API requests grows.

6.1 Evaluation Methodology

The evaluation process focuses on two different end points with the middleware and the other without it, both the API have the same business logic. The Load was generated using k6 Grafana scripts which helped to execute repeatable tests with controllable traffic pattern which also helped to export the results to automatically to Grafana dashboards. For each test run the user can select the number of API request from the test simulation page, for each test run the script will automatically send fixed number of HTTP requests to the API and key performance metrics like average latency, max latency, number of positive (HTTP 200) and negative requests (HTTP 403).

To understand how this middleware behaves when the number of API requests grow five different test runs were executed with 1000, 2000, 3000, 4000, 5000 total requests were made respectively. Each run has a 50 percent of compliant requests and 50 percent of non compliant requests. Compliant requests were the request that follows the tenant's configured rules. Non compliant are the request that deliberately violate at least one of the violation so that it will be denied by the middleware. This design helps to evaluate the correctness of the middleware and also ensure that both positive and negative paths are tested for every volume.

6.2 Results Overview

Table 3 summaries the results that were collected from five test runs which are extracted from Grafana dashboard. This test results represents actual telemetry and not synthetic projections.

Table 3: Latency and Request Count Performance Comparison: Middleware vs. Baseline API

Req.	**Avg Lat. Mid**	**Avg Lat. No-Mid**	**Max Mid**	**Max No-Mid**	**Positive**	**Negative**
1000	49.0 ms	47.7 ms	78.3 ms	60.2 ms	506	494
2000	67.3 ms	67.0 ms	73.8 ms	72.5 ms	1002	998
3000	66.0 ms	68.4 ms	67.9 ms	72.0 ms	1456	1544
4000	66.5 ms	67.6 ms	74.8 ms	74.4 ms	2000	2000
5000	33.1 ms	35.3 ms	42.1 ms	65.4 ms	2508	2492

All five tests indicate that average latency for all the test runs stays almost consistent which ranges from 33.1 to 67.3 ms, which also as an overall average of about 56.4 ms. The 5000 test run has a low latency of 33.1 ms which indicates that the system works very well even in heavy load. The maximum latency in all the test runs stayed at 42.1 ms and 78.3 ms which indicates that there were no sudden spike or delay which shows that the system was overloaded. As shown in table 3 these consistent rules proves that middleware does not have much performance overhead. The middleware performs different tasks like collecting request context, checking zero trust roles, calculating the trust score and making access decisions based on trust score but this add only a small amount of time to each request. As shown in Fig. 5 which displays the Grafana dashboard for 5000 test run which shows the latency, Http status code breakdown and request count metrics which are directly collected from K6 script execution and github workflow. The Fig.5

confirms that Table 3 metrics were collected from actual telemetry which demonstrates the stability of the middleware across different loads.

6.3 Security Effectiveness

The success and deny rates provides an idea about how accurately the middleware enforces the zero trust policies. The expected output for the correct implementation were 50 percent of positive (HTTP 200) and negative (HTTP 403) requests. The results in table 3 shows exceptional alignment with this theoretical distribution with success rates which ranges from 48.5 percent to 50.6 percent and deny rates ranges from 49.4 percent to 51.8 percent. This consistency across all different test runs with varying load requests shows that trust score calculation and access decisions behave reliably even as the request volume grows.

The negligible variation from the expected 50:50 split is entirely consistent from randomized test data. No systematic bias were observed towards either positive or negative requests. Importantly across all 10,000 test requests no compliant request was categorized as unauthorized and no non compliant request was allowed. This absence of false positives and false negatives shows that implementation of rule parsing, score calculation and threshold comparison is functionally correct for the tested requests. which denotes the reliability of the zero trust middleware.

Furthermore validation from Grafana cloud insights as in Fig. 6 confirms this behavior. During the 10,000 test API requests the baseline endpoint without the middlware permitted all 5,000 requests, whereas the API end point that enforces the middleware results in an exact 50:50 split with 2,500 allowed and 2,500 denied requests. This quantitatively demonstrates the middleware's active security enforcement.

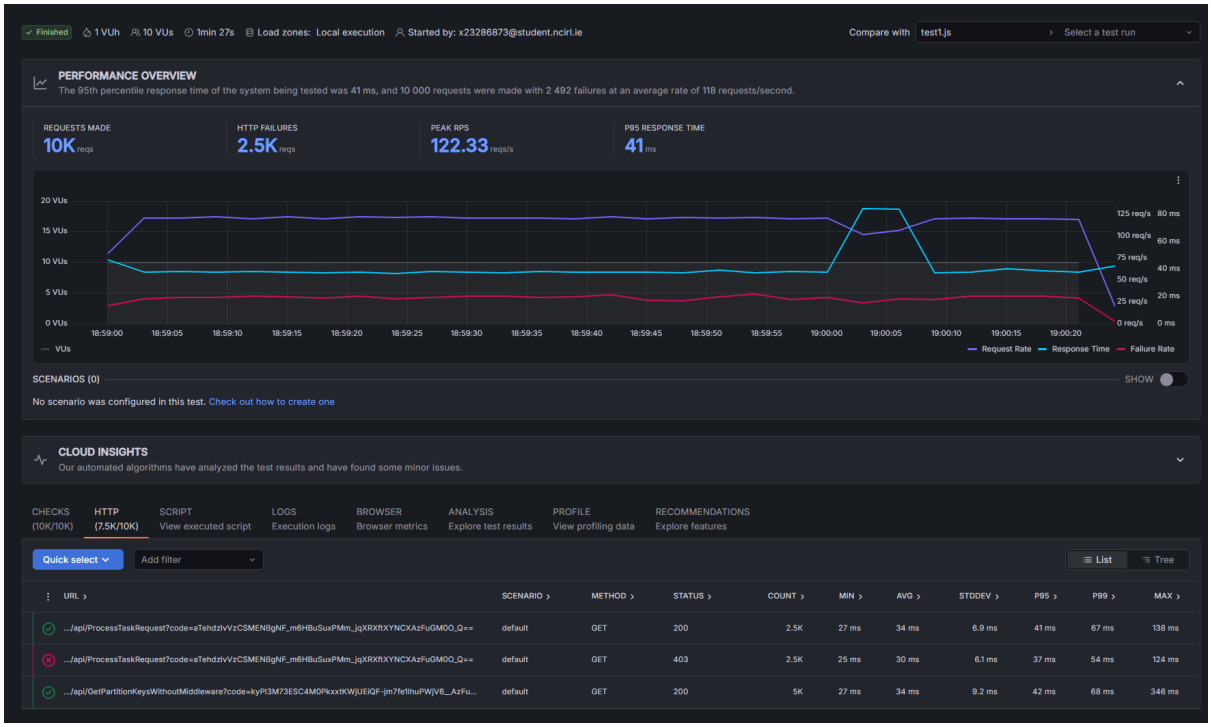

Figure 6: Grafana Cloud Dashboard for API Request Counts.

6.4 Performance Discussion

From a performance perspective, average latency is below 100 to 200 ms range that is acceptable for interactive workloads. The variation in average latency suggests natural fluctuation due to factors such as network issues, Azure functions cold starts rather than the system's degradation from middleware itself.

The maximum latency values such as 42.1 to 78.3 ms with no outliers or runaway increases confirms that the system can handle peak bursts without any saturation. Notably the 5000 requests shows lowest average latency 33.1ms and lowest maximum latency of 42.1ms which suggests efficient utilization of resources at high load levels. Throughput which can be derived from request counts and test durations always remained stable. The consistent 50:50 split of positive and negative tests show that middleware does not become a bottle neck as traffic increases. Additionally as shown in Fig. 6 the middleware maintained a 95th percentile (P95) response time of 41ms even when processing a peak throughput of 122 requests per second which indicates that 95 percent of all requests were completed in 41 ms or less.

The comparable difference between the request volume shows that the middleware can scale efficiently even as the request volume increases. The Azure Functions environment appears to handle the concurrent requests appropriately which adds minimal overhead to the total request response cycle. These results shows that the additional security checks are compatible with the performance requirements of a multitenant API.

6.5 Discussion

The evaluation section successfully demonstrates that the ZTA middleware successfully enforced tenant specific security policies with minimal performance over head. Around 1 to 3 percent when compared to the response of the base line API. Average latency of 56.4 ms remained below the industry standard of 100 to 200 ms threshold. The 95th percentile response time is 41 ms under peak throughput of 122 requests per second confirms that the system can scale efficiently. The near perfect split of 50:50 of positive and negative requests validates the correctness of the trust score calculations.

6.6 Limitations

Testing was limited to a single end point with a maximum of 5,000 requests for each test run. The artificial 50:50 split of compliant and non compliant request does not reflect real world distribution which can typically have more than 95 percent of compliant request. All the tests were executed from a single geographical location and end to end latency may include network issue, Azure function cold start. Additionally the system does not assess the system behavior under high volume data growth scenarios as tenant level data can accumulate over time. The overhead of policy evaluation may increase due to increase of large rule sets and increased database query complexity, which can impact performance at some scale in a production environment.

7 Conclusion and Future Work

This research project successfully designed, implemented and evaluated a Zero Trust .Net core middleware to mitigate multitenant data leakage focusing on Azure Functions.

This implementation proves that attribute based access control (ABAC) policies can be implemented at the API data access layer without significant performance degradation. Some of the key findings of this project is the middleware enforces tenant specific rules with near prefect accuracy which ensures that there are no false positives or false negatives. The system has very low latency overhead with average response time of 56.4 ms and P95 latency of 41 ms which is well within the acceptable thresholds of interactive work loads. The middleware also scales efficiently over increasing load maintaining stable throughput across increasing request volumes ranging from 1000 to 5000 requests.

By moving security from perimeter to data layer organizations can decide which tenants can access which data resources within a shared infrastructure. The Azure function API deployments offer better elasticity and cost efficiency. This Zero trust middleware offers light weight and flexibility for custom security policies without the need of much code changes. This project demonstrates Zero trust not just based on theoretical principles but also in a practical implementation for .Net core cloud native applications.

Some of the future work that can we carried on in this research project is distributed caching using Redis/Memcached to eliminate the repeated rule violations which can reduce the response time by 20 to 40 percent. Enhanced policy can implement machine learning models can be implemented to validate every request and detect any violations. Finally a real world SaaS deployment that implements this middle ware would provide invaluable insights to understand any edge cases, scaling challenges which may not be addressed in simulated performance or load testing.

References

Anuganti, C. (n.d.). Scalable multi-tenant data lakes with zero trust security architecture.

Balasubramanian, P. N. (n.d.). Security pattern catalog for azure saas applications.

Chippagiri, S. (2025). A study of cloud security frameworks for safeguarding multi-tenant cloud architectures, *International Journal of Computer Applications* **975**: 8887.

Denis, R., Kumar, B. and Manivannan, T. (2025). Ai-driven automated threat hunting for secure and scalable private clouds: A comprehensive review: Ai-driven automated threat hunting for secure and scalable private clouds: A comprehensive review, *SGS-Engineering & Sciences* **1**(4).

Hayat, M., Islam, S. and Hossain, M. (2024). Securing the cloud infrastructure: Investigating multi-tenancy challenges, *Modern Solutions and Future Research Opportunities* .

Krishnappa, M. S., Veerapaneni, P. K., Harve, B. M., Jayaram, V., Bidkar, D. M., Mehta, G. and Yogeshappa, V. G. (2024). Cybersecurity in the cloud era: Protecting virtualized environments against evolving threats, pp. 1049–1054.

kumar Polinati, A. (2025). Hybrid cloud security: Balancing performance, cost, and compliance in multi-cloud deployments.

Mavroudis, V. (2024). Zero-trust network access (ztna), *arXiv preprint arXiv:2410.20611* .

Mehata, M. (2025). *Dynamic zero trust access control: Fortifying security in mobile-cloud environment*, Master's thesis, Youngstown State University.

Müller, A. K. (2025). Integrating environmental pollutant analytics and cancer detection into an ai-driven rural health cloud with zero-trust security and lddr optimization, *International Journal of Engineering & Extended Technologies Research (IJEETR)* **7**(6): 10937–10941.

Obaze, C. A., Onwuegbuzie, I. U. and Onwujei, A. I. (2025). Adaptive intelligence for zero trust (aizt): An ai-driven conceptual framework for proactive cloud cybersecurity infrastructure, *Tech-Sphere Journal for Pure and Applied Sciences* **2**(1): 42–56.

OWASP Foundation (n.d.). Owasp cloud tenant isolation, `https://owasp.org/www-project-cloud-tenant-isolation`. Accessed: 11 December 2025.

Ponmalar, A., Pandiarajan, R., Sudha, I., Ramesh, P. and Jagannathan, J. (2024). Securing multi-tenant cloud environments with fully homomorphically encrypted secure multiparty computation, pp. 1–6.

Rehan, H. (2022). Zero-trust architecture for securing multi-cloud environments.

Steingartner, W., Galinec, D. and Zebić, V. (2024). Challenges of application programming interfaces security: A conceptual model in the changing cyber defense environment and zero trust architecture, pp. 372–379.

Vesga, E. (2024). Protecting privileged access to cloud computing's saas services.

Viswanathan, J., Kumar, S. U. et al. (2024). Zero trust security for web applications in microservice-based environments, pp. 488–494.

Wang, W., Sadjadi, S. M., Rishe, N. and Mahara, A. (2024). Applying transparent shaping for zero trust architecture implementation in aws: A case study, *arXiv preprint arXiv:2405.01412* .